autogluon_each_location

October 18, 2023

[1]:

```python
# config

label = 'y'
metric = 'mean_absolute_error'
time_limit = None
presets = 'best_quality'

do_drop_ds = True
# hour, dayofweek, dayofmonth, month, year
use_dt_attrs = []#["hour", "year"]
use_estimated_diff_attr = False
use_is_estimated_attr = True

use_groups = False
n_groups = 8

auto_stack = False
num_stack_levels = 0
num_bag_folds = 8
num_bag_sets = 20

use_tune_data = True
use_test_data = True
tune_and_test_length = 24*30*6 # 3 months from end
holdout_frac = None
use_bag_holdout = True # Enable this if there is a large gap between score_val␣
 ↪and score_test in stack models.

sample_weight = None#'sample_weight' #None
weight_evaluation = False
sample_weight_estimated = 1

run_analysis = False
```

[2]:

```python
import pandas as pd
import numpy as np
```

```python
import warnings
warnings.filterwarnings("ignore")


def feature_engineering(X):
    # shift all columns with "1h" in them by 1 hour, so that for index 16:00,␣
↪we have the values from 17:00
    # but only for the columns with "1h" in the name
    #X_shifted = X.filter(regex="\dh").shift(-1, axis=1)
    #print(f"Number of columns with 1h in name: {X_shifted.columns}")


    columns = ['clear_sky_energy_1h:J', 'diffuse_rad_1h:J', 'direct_rad_1h:J',
       'fresh_snow_12h:cm', 'fresh_snow_1h:cm', 'fresh_snow_24h:cm',
       'fresh_snow_3h:cm', 'fresh_snow_6h:cm']

    X_shifted = X[X.index.minute==0][columns].copy()
    # loop through all rows and check if index + 1 hour is in the index, if so␣
↪get that value, else nan
    count1 = 0
    count2 = 0
    for i in range(len(X_shifted)):
        if X_shifted.index[i] + pd.Timedelta('1 hour') in X.index:
            count1 += 1
            X_shifted.iloc[i] = X.loc[X_shifted.index[i] + pd.Timedelta('1␣
↪hour')][columns]
        else:
            count2 += 1
            X_shifted.iloc[i] = np.nan

    print("COUNT1", count1)
    print("COUNT2", count2)

    X_old_unshifted = X[X.index.minute==0][columns]
    # rename X_old_unshifted columns to have _not_shifted at the end
    X_old_unshifted.columns = [f"{col}_not_shifted" for col in X_old_unshifted.
↪columns]

    # put the shifted columns back into the original dataframe
    #X[columns] = X_shifted[columns]



    date_calc = None
    if "date_calc" in X.columns:
```

```
        date_calc = X[X.index.minute == 0]['date_calc']

    # resample to hourly
    print("index: ", X.index[0])
    X = X.resample('H').mean()
    print("index AFTER: ", X.index[0])

    X[columns] = X_shifted[columns]
    #X[X_old_unshifted.columns] = X_old_unshifted

    if date_calc is not None:
        X['date_calc'] = date_calc

    return X




def fix_X(X, name):
    # Convert 'date_forecast' to datetime format and replace original column␣
 ↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)


    X = feature_engineering(X)

    return X



def handle_features(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_X(X_train_observed, "X_train_observed")
    X_train_estimated = fix_X(X_train_estimated, "X_train_estimated")
    X_test = fix_X(X_test, "X_test")


    if weight_evaluation:
        # add sample weights, which are 1 for observed and 3 for estimated
        X_train_observed["sample_weight"] = 1
        X_train_estimated["sample_weight"] = sample_weight_estimated
        X_test["sample_weight"] = sample_weight_estimated


    y_train['ds'] = pd.to_datetime(y_train['time'])
```

```python
    y_train.drop(columns=['time'], inplace=True)
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train




def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,␣
 ↪location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =␣
 ↪handle_features(X_train_observed, X_train_estimated, X_test, y_train)

    if use_estimated_diff_attr:
        X_train_observed["estimated_diff_hours"] = 0
        X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index -␣
 ↪pd.to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
        X_test["estimated_diff_hours"] = (X_test.index - pd.
 ↪to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

        X_train_estimated["estimated_diff_hours"] =␣
 ↪X_train_estimated["estimated_diff_hours"].astype('int64')
        # the filled once will get dropped later anyways, when we drop y nans
        X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].
 ↪fillna(-50).astype('int64')

    if use_is_estimated_attr:
        X_train_observed["is_estimated"] = 0
        X_train_estimated["is_estimated"] = 1
        X_test["is_estimated"] = 1

    # drop date_calc
    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)


    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])


    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)
```

```python
    X_train = pd.merge(X_train, y_train, how="inner", left_index=True,␣
↪right_index=True)

    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")


    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed= pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,␣
↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)
```

```
Processing location A…
COUNT1 29667
COUNT2 1
index:  2019-06-02 22:00:00
index AFTER:  2019-06-02 22:00:00
```

```
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 0
Processing location B…
COUNT1 29232
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 4
Processing location C…
COUNT1 29206
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 6059
```

1 Feature enginering

[3]:
```
import numpy as np
import pandas as pd

X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],␣
↪inplace=True)
```

```python
for attr in use_dt_attrs:
    X_train[attr] = getattr(X_train.index, attr)
    X_test[attr] = getattr(X_test.index, attr)

print(X_train.head())




if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique()  # Assuming 'location' is the name␣
 ↪of the column representing locations

    grouped_dfs = []  # To store data frames split by location

    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()

        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups),␣
 ↪repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())




to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:ms",␣
 ↪"dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:cm",␣
 ↪"wind_speed_u_10m:ms", "wind_speed_v_10m:ms"]

X_train.drop(columns=to_drop, inplace=True)
```

```
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)
```

```
                     absolute_humidity_2m:gm3  air_density_2m:kgm3  \
ds
2019-06-02 22:00:00                     7.700              1.22825
2019-06-02 23:00:00                     7.700              1.22350
2019-06-03 00:00:00                     7.875              1.21975
2019-06-03 01:00:00                     8.425              1.21800
2019-06-03 02:00:00                     8.950              1.21800

                     ceiling_height_agl:m  clear_sky_energy_1h:J  \
ds
2019-06-02 22:00:00           1728.949951               0.000000
2019-06-02 23:00:00           1689.824951               0.000000
2019-06-03 00:00:00           1563.224976               0.000000
2019-06-03 01:00:00           1283.425049            6546.899902
2019-06-03 02:00:00           1003.500000          102225.898438

                     clear_sky_rad:W  cloud_base_agl:m  dew_or_rime:idx  \
ds
2019-06-02 22:00:00             0.00       1728.949951              0.0
2019-06-02 23:00:00             0.00       1689.824951              0.0
2019-06-03 00:00:00             0.00       1563.224976              0.0
2019-06-03 01:00:00             0.75       1283.425049              0.0
2019-06-03 02:00:00            23.10       1003.500000              0.0

                     dew_point_2m:K  diffuse_rad:W  diffuse_rad_1h:J  …  \
ds                                                                    …
2019-06-02 22:00:00      280.299988          0.000          0.000000  …
2019-06-02 23:00:00      280.299988          0.000          0.000000  …
2019-06-03 00:00:00      280.649994          0.000          0.000000  …
2019-06-03 01:00:00      281.674988          0.300       7743.299805  …
2019-06-03 02:00:00      282.500000         11.975      60137.601562  …

                     t_1000hPa:K  total_cloud_cover:p  visibility:m  \
ds
2019-06-02 22:00:00   286.225006           100.000000  40386.476562
2019-06-02 23:00:00   286.899994           100.000000  33770.648438
2019-06-03 00:00:00   286.950012           100.000000  13595.500000
2019-06-03 01:00:00   286.750000           100.000000   2321.850098
2019-06-03 02:00:00   286.450012            99.224998  11634.799805

                     wind_speed_10m:ms  wind_speed_u_10m:ms  \
ds
2019-06-02 22:00:00              3.600               -3.575
```

```
2019-06-02 23:00:00              3.350             -3.350
2019-06-03 00:00:00              3.050             -2.950
2019-06-03 01:00:00              2.725             -2.600
2019-06-03 02:00:00              2.550             -2.350

                     wind_speed_v_10m:ms  wind_speed_w_1000hPa:ms  \
ds
2019-06-02 22:00:00               -0.500                      0.0
2019-06-02 23:00:00                0.275                      0.0
2019-06-03 00:00:00                0.750                      0.0
2019-06-03 01:00:00                0.875                      0.0
2019-06-03 02:00:00                0.925                      0.0

                     is_estimated      y  location
ds
2019-06-02 22:00:00             0   0.00         A
2019-06-02 23:00:00             0   0.00         A
2019-06-03 00:00:00             0   0.00         A
2019-06-03 01:00:00             0   0.00         A
2019-06-03 02:00:00             0  19.36         A

[5 rows x 48 columns]
```

[4]:

```
from autogluon.tabular import TabularDataset, TabularPredictor
from autogluon.timeseries import TimeSeriesDataFrame
import numpy as np
train_data = TabularDataset('X_train_raw.csv')
# set group column of train_data be increasing from 0 to 7 based on time, the
 ↪first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
train_data['ds'] = pd.to_datetime(train_data['ds'])
train_data = train_data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:
#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())


# get end date of train data and subtract 3 months
#split_time = pd.to_datetime(train_data["ds"]).max() - pd.
 ↪Timedelta(hours=tune_and_test_length)
# 2022-10-28 22:00:00
split_time = pd.to_datetime("2022-10-28 22:00:00")
train_set = TabularDataset(train_data[train_data["ds"] < split_time])
test_set = TabularDataset(train_data[train_data["ds"] >= split_time])
if use_groups:
    test_set = test_set.drop(columns=['group'])
```

```python
if do_drop_ds:
    train_set = train_set.drop(columns=['ds'])
    test_set = test_set.drop(columns=['ds'])
    train_data = train_data.drop(columns=['ds'])


def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] /
 ↪loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

tuning_data = None
if use_tune_data:
    train_data = train_set
    if use_test_data:
        # split test_set in half, use first half for tuning
        tuning_data, test_data = [], []
        for loc in locations:
            loc_test_set = test_set[test_set["location"] == loc]
            # randomly shuffle the loc_test_set
            loc_tuning_data, loc_test_data = pd.DataFrame(), pd.DataFrame()
            for i in range(200):
                # get a part of the test set corresponding to i/100th part of
 ↪the test set and shuffle
                num_bins = len(loc_test_set) // 200
                # set seed to i so that we get the same shuffle every time
                np.random.seed(i)
                current_bin = loc_test_set.iloc[i*num_bins:min((i+1)*num_bins,
 ↪len(loc_test_set))].sample(frac=1)
                loc_tuning_data = pd.concat([loc_tuning_data, current_bin.iloc[:
 ↪len(current_bin)//2]])
                loc_test_data = pd.concat([loc_test_data, current_bin.
 ↪iloc[len(current_bin)//2:]])


            tuning_data.append(loc_tuning_data)
            test_data.append(loc_test_data)
        tuning_data = pd.concat(tuning_data)
        test_data = pd.concat(test_data)
        print("Shapes of tuning and test", tuning_data.shape[0], test_data.
 ↪shape[0], tuning_data.shape[0] + test_data.shape[0])

    else:
```

```
        tuning_data = test_set
        print("Shape of tuning", tuning_data.shape[0])

    # ensure sample weights for your tuning data sum to the number of rows in␣
 ↪the tuning data.
    if weight_evaluation:
        tuning_data = normalize_sample_weights_per_location(tuning_data)


else:
    if use_test_data:
        train_data = train_set
        test_data = test_set
        print("Shape of test", test_data.shape[0])

# ensure sample weights for your training (or tuning) data sum to the number of␣
 ↪rows in the training (or tuning) data.
if weight_evaluation:
    train_data = normalize_sample_weights_per_location(train_data)
    if use_test_data:
        test_data = normalize_sample_weights_per_location(test_data)


train_data = TabularDataset(train_data)
if use_tune_data:
    tuning_data = TabularDataset(tuning_data)
if use_test_data:
    test_data = TabularDataset(test_data)
```

```
Shapes of tuning and test 5000 5400 10400
```

[5]:
```
if run_analysis:
    import autogluon.eda.auto as auto
    auto.dataset_overview(train_data=train_data, test_data=test_data,␣
 ↪label="y", sample=None)
```

[6]:
```
if run_analysis:
    auto.target_analysis(train_data=train_data, label="y", sample=None)
```

2 Starting

[7]:
```
import os


# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for␣
 ↪filename in os.listdir('submissions') if "submission" in filename]))
```

```
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'

hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)
```

```
Last submission number: 93
Now creating submission number: 94
New filename: submission_94
```

[8]:
```
predictors = [None, None, None]
```

[9]:
```
def fit_predictor_for_location(loc):
    print(f"Training model for location {loc}...")
    # sum of sample weights for this location, and number of rows, for both␣
 ↪train and tune data and test data
    if weight_evaluation:
        print("Train data sample weight sum:",␣
 ↪train_data[train_data["location"] == loc]["sample_weight"].sum())
        print("Train data number of rows:", train_data[train_data["location"]␣
 ↪== loc].shape[0])
        if use_tune_data:
            print("Tune data sample weight sum:",␣
 ↪tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
            print("Tune data number of rows:",␣
 ↪tuning_data[tuning_data["location"] == loc].shape[0])
        if use_test_data:
            print("Test data sample weight sum:",␣
 ↪test_data[test_data["location"] == loc]["sample_weight"].sum())
            print("Test data number of rows:", test_data[test_data["location"]␣
 ↪== loc].shape[0])
    predictor = TabularPredictor(
        label=label,
        eval_metric=metric,
        path=f"AutogluonModels/{new_filename}_{loc}",
        # sample_weight=sample_weight,
        # weight_evaluation=weight_evaluation,
        # groups="group" if use_groups else None,
    ).fit(
        train_data=train_data[train_data["location"] == loc],
        time_limit=time_limit,
```

```
        presets=presets,
        num_stack_levels=num_stack_levels,
        num_bag_folds=num_bag_folds if not use_groups else 2,# just put␣
↪somethin, will be overwritten anyways
        num_bag_sets=num_bag_sets,
        tuning_data=tuning_data[tuning_data["location"] == loc].
↪reset_index(drop=True) if use_tune_data else None,
        use_bag_holdout=use_bag_holdout,
        # holdout_frac=holdout_frac,
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc]#.
↪drop(columns=["sample_weight"])
        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)
```

```
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=1, num_bag_folds=8,
num_bag_sets=1
Beginning AutoGluon training …
AutoGluon will save models to "AutogluonModels/submission_94_A/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   236.41 GB / 315.93 GB (74.8%)
Train Data Rows:    29667
Train Data Columns: 38
Tuning Data Rows:    2000
Tuning Data Columns: 38
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (5733.42, 0.0, 674.14552,
1195.53172)
        If 'regression' is not the correct problem_type, please manually specify
```

```
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    132284.33 MB
        Train Data (Original)  Memory Usage: 11.21 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…

Training model for location A…

        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 35 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 35 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        36 features in original data used to generate 36 features in processed
data.
        Train Data (Processed) Memory Usage: 8.9 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.16s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
```

```
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
AutoGluon will fit 2 stack levels (L1 to L2) …
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 …
        -133.6185        = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.39s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 …
        -133.2862        = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.36s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -109.6322        = Validation score   (-mean_absolute_error)
        30.9s    = Training   runtime
        12.85s   = Validation runtime
Fitting model: LightGBM_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -112.566         = Validation score   (-mean_absolute_error)
        30.13s   = Training   runtime
        14.93s   = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 …
        -115.8985        = Validation score   (-mean_absolute_error)
        8.42s    = Training   runtime
```

```
        1.14s    = Validation runtime
Fitting model: CatBoost_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -119.1588        = Validation score   (-mean_absolute_error)
        200.7s   = Training   runtime
        0.12s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 …
        -116.9235        = Validation score   (-mean_absolute_error)
        1.69s    = Training   runtime
        1.13s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -123.2376        = Validation score   (-mean_absolute_error)
        37.05s   = Training   runtime
        0.49s    = Validation runtime
Fitting model: XGBoost_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -113.5542        = Validation score   (-mean_absolute_error)
        51.37s   = Training   runtime
        2.68s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -105.7007        = Validation score   (-mean_absolute_error)
        94.72s   = Training   runtime
        0.37s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -109.2278        = Validation score   (-mean_absolute_error)
        100.24s  = Training   runtime
        21.28s   = Validation runtime
Fitting model: WeightedEnsemble_L2 …
        -104.6917        = Validation score   (-mean_absolute_error)
        0.43s    = Training   runtime
        0.0s     = Validation runtime
Fitting 9 L2 models …
Fitting model: LightGBMXT_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -105.6881        = Validation score   (-mean_absolute_error)
        3.04s    = Training   runtime
        0.17s    = Validation runtime
Fitting model: LightGBM_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
```

```
ParallelLocalFoldFittingStrategy
	-107.9129	 = Validation score   (-mean_absolute_error)
	1.89s	 = Training   runtime
	0.08s	 = Validation runtime
Fitting model: RandomForestMSE_BAG_L2 …
	-109.2965	 = Validation score   (-mean_absolute_error)
	13.28s	 = Training   runtime
	1.22s	 = Validation runtime
Fitting model: CatBoost_BAG_L2 …
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-107.5346	 = Validation score   (-mean_absolute_error)
	5.29s	 = Training   runtime
	0.05s	 = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L2 …
	-107.6613	 = Validation score   (-mean_absolute_error)
	2.19s	 = Training   runtime
	1.19s	 = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L2 …
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-110.219	 = Validation score   (-mean_absolute_error)
	37.09s	 = Training   runtime
	0.49s	 = Validation runtime
Fitting model: XGBoost_BAG_L2 …
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-107.5087	 = Validation score   (-mean_absolute_error)
	2.85s	 = Training   runtime
	0.11s	 = Validation runtime
Fitting model: NeuralNetTorch_BAG_L2 …
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-104.6545	 = Validation score   (-mean_absolute_error)
	53.87s	 = Training   runtime
	0.5s	 = Validation runtime
Fitting model: LightGBMLarge_BAG_L2 …
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-107.8037	 = Validation score   (-mean_absolute_error)
	5.25s	 = Training   runtime
	0.18s	 = Validation runtime
Fitting model: WeightedEnsemble_L3 …
	-104.567	 = Validation score   (-mean_absolute_error)
	0.35s	 = Training   runtime
	0.0s	 = Validation runtime
AutoGluon training complete, total runtime = 735.71s … Best model:
"WeightedEnsemble_L3"
```

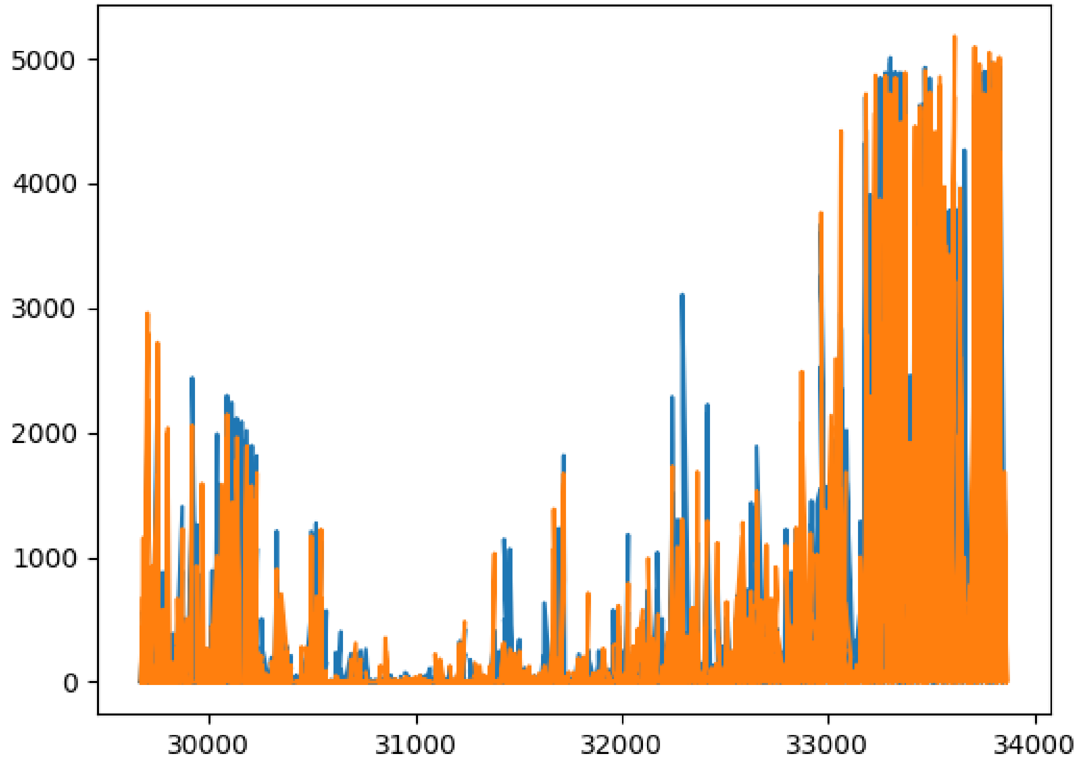
```
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_94_A/")
Evaluation: mean_absolute_error on test data: -108.60384533057054
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -108.60384533057054,
    "root_mean_squared_error": -305.6816918188323,
    "mean_squared_error": -93441.29671322356,
    "r2": 0.8805259503688295,
    "pearsonr": 0.9385599911845793,
    "median_absolute_error": -0.41422155499458313
}

Evaluation on test data:
-108.60384533057054
```

[10]:

```python
leaderboards = []
if use_test_data:
    lb = predictors[0].leaderboard(test_data[test_data["location"] == loc])
    lb["location"] = loc
    leaderboards.append(lb)
    test_data[test_data["location"] == loc]["y"].plot()
    if use_tune_data:
        tuning_data[tuning_data["location"] == loc]["y"].plot()
```

```
                     model  score_test   score_val  pred_time_test
pred_time_val    fit_time  pred_time_test_marginal  pred_time_val_marginal
fit_time_marginal  stack_level  can_infer  fit_order
0      WeightedEnsemble_L2 -106.464945 -104.691735        8.610645
34.861226  226.312503                 0.004399                0.000636
0.432885            2       True          12
1    NeuralNetTorch_BAG_L1 -106.483024 -105.700704        0.246203
0.366351   94.717332                 0.246203                0.366351
94.717332            1       True          10
2      WeightedEnsemble_L3 -108.603845 -104.566951       14.854486
56.399232  612.512369                 0.002249                0.000729
0.347646            3       True          22
3    NeuralNetTorch_BAG_L2 -109.282383 -104.654502       14.784469
56.230217  609.129228                 0.372648                0.495219
53.866686            2       True          20
4           XGBoost_BAG_L2 -111.336104 -107.508695       14.499334
55.844983  558.110395                 0.087513                0.109986
2.847853            2       True          19
5         LightGBMXT_BAG_L2 -111.378388 -105.688058       14.479589
55.903284  558.298036                 0.067768                0.168286
3.035494            2       True          13
6     LightGBMLarge_BAG_L2 -111.447478 -107.803747       14.547423
```

```
55.910179  560.507911                 0.135602                 0.175181
5.245369            2       True         21
7   RandomForestMSE_BAG_L2 -111.668660 -109.296451       15.014299
56.959545  568.539886                 0.602477                 1.224547
13.277343            2       True         15
8           CatBoost_BAG_L2 -111.691459 -107.534587       14.454121
55.787292  560.553231                 0.042300                 0.052295
5.290688            2       True         16
9     ExtraTreesMSE_BAG_L2 -111.778220 -107.661283       15.008604
56.926856  557.452750                 0.596782                 1.191859
2.190207            2       True         17
10          LightGBM_BAG_L2 -112.451150 -107.912871       14.449175
55.818054  557.155247                 0.037354                 0.083057
1.892704            2       True         14
11     LightGBMLarge_BAG_L1 -113.161480 -109.227811        4.858676
21.281557  100.236939                 4.858676                21.281557
100.236939            1       True         11
12  NeuralNetFastAI_BAG_L2 -113.675624 -110.219003       15.063398
56.223438  592.356961                 0.651576                 0.488440
37.094419            2       True         18
13          LightGBM_BAG_L1 -116.417802 -112.565988        2.965765
14.931696   30.125217                 2.965765                14.931696
30.125217            1       True          4
14        LightGBMXT_BAG_L1 -117.168060 -109.632222        3.465628
12.850172   30.898239                 3.465628                12.850172
30.898239            1       True          3
15           XGBoost_BAG_L1 -117.911577 -113.554171        0.935745
2.682815   51.367466                 0.935745                 2.682815
51.367466            1       True          9
16     ExtraTreesMSE_BAG_L1 -118.563553 -116.923479        0.556412
1.130832    1.693071                 0.556412                 1.130832
1.693071            1       True          7
17  RandomForestMSE_BAG_L1 -119.504864 -115.898548        0.551370
1.135428    8.415134                 0.551370                 1.135428
8.415134            1       True          5
18          CatBoost_BAG_L1 -125.072803 -119.158813        0.103171
0.117834  200.702585                 0.103171                 0.117834
200.702585            1       True          6
19  NeuralNetFastAI_BAG_L1 -129.000628 -123.237616        0.644522
0.485936   37.052310                 0.644522                 0.485936
37.052310            1       True          8
20   KNeighborsDist_BAG_L1 -134.896380 -133.286181        0.035738
0.362511    0.027107                 0.035738                 0.362511
0.027107            1       True          2
21   KNeighborsUnif_BAG_L1 -135.049170 -133.618518        0.048592
0.389867    0.027141                 0.048592                 0.389867
0.027141            1       True          1
```



[11]:
```
loc = "B"
predictors[1] = fit_predictor_for_location(loc)
```

```
Training model for location B…

Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=1, num_bag_folds=8,
num_bag_sets=1
Beginning AutoGluon training …
AutoGluon will save models to "AutogluonModels/submission_94_B/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   233.15 GB / 315.93 GB (73.8%)
Train Data Rows:    29218
Train Data Columns: 38
Tuning Data Rows:    1600
Tuning Data Columns: 38
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
```

```
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (1152.3, 0.0, 102.58516, 198.99359)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    130212.9 MB
        Train Data (Original)  Memory Usage: 10.91 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 35 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 35 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        36 features in original data used to generate 36 features in processed
data.
        Train Data (Processed) Memory Usage: 8.66 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.17s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
```

```
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
AutoGluon will fit 2 stack levels (L1 to L2) …
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 …
        -25.9296         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.37s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 …
        -26.2267         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.38s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -22.8699         = Validation score   (-mean_absolute_error)
        30.66s   = Training   runtime
        13.9s    = Validation runtime
Fitting model: LightGBM_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -21.5638         = Validation score   (-mean_absolute_error)
        32.67s   = Training   runtime
        19.25s   = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 …
        -19.0379         = Validation score   (-mean_absolute_error)
```

```
        9.39s    = Training   runtime
        1.09s    = Validation runtime
Fitting model: CatBoost_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -22.5788         = Validation score   (-mean_absolute_error)
        197.16s  = Training   runtime
        0.1s     = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 …
        -19.1835         = Validation score   (-mean_absolute_error)
        1.62s    = Training   runtime
        1.11s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -22.355  = Validation score   (-mean_absolute_error)
        35.5s    = Training   runtime
        0.44s    = Validation runtime
Fitting model: XGBoost_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -21.4734         = Validation score   (-mean_absolute_error)
        92.57s   = Training   runtime
        20.57s   = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -19.4201         = Validation score   (-mean_absolute_error)
        160.61s  = Training   runtime
        0.37s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -19.942  = Validation score   (-mean_absolute_error)
        102.46s  = Training   runtime
        22.49s   = Validation runtime
Fitting model: WeightedEnsemble_L2 …
        -18.2596         = Validation score   (-mean_absolute_error)
        0.41s    = Training   runtime
        0.0s     = Validation runtime
Fitting 9 L2 models …
Fitting model: LightGBMXT_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -20.5565         = Validation score   (-mean_absolute_error)
        4.09s    = Training   runtime
        0.3s     = Validation runtime
Fitting model: LightGBM_BAG_L2 …
```

```
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -20.6628         = Validation score   (-mean_absolute_error)
        2.23s    = Training   runtime
        0.12s    = Validation runtime
Fitting model: RandomForestMSE_BAG_L2 …
        -20.7892         = Validation score   (-mean_absolute_error)
        13.43s   = Training   runtime
        1.12s    = Validation runtime
Fitting model: CatBoost_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -20.3994         = Validation score   (-mean_absolute_error)
        9.0s     = Training   runtime
        0.06s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L2 …
        -20.5017         = Validation score   (-mean_absolute_error)
        2.02s    = Training   runtime
        1.14s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -20.9894         = Validation score   (-mean_absolute_error)
        36.19s   = Training   runtime
        0.49s    = Validation runtime
Fitting model: XGBoost_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -20.7225         = Validation score   (-mean_absolute_error)
        2.93s    = Training   runtime
        0.11s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -20.5158         = Validation score   (-mean_absolute_error)
        100.1s   = Training   runtime
        0.54s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -20.5379         = Validation score   (-mean_absolute_error)
        67.01s   = Training   runtime
        1.66s    = Validation runtime
Fitting model: WeightedEnsemble_L3 …
        -20.2692         = Validation score   (-mean_absolute_error)
        0.34s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 957.88s … Best model:
```

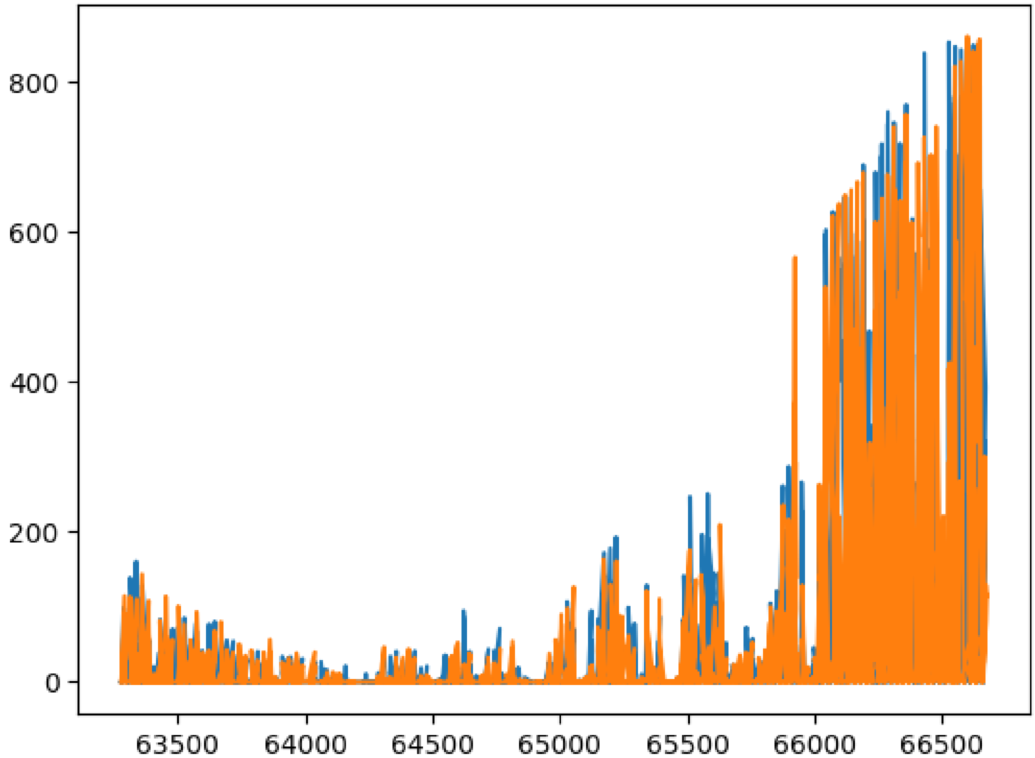
```
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_94_B/")
Evaluation: mean_absolute_error on test data: -19.784612134597683
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -19.784612134597683,
    "root_mean_squared_error": -59.48722932746857,
    "mean_squared_error": -3538.7304530588362,
    "r2": 0.8091036613883708,
    "pearsonr": 0.9210702081720571,
    "median_absolute_error": -1.9378460291816468
}

Evaluation on test data:
-19.784612134597683
```

[12]:

```
if use_test_data:
    lb = predictors[1].leaderboard(test_data[test_data["location"] == loc])
    test_data[test_data["location"] == loc]["y"].plot()
    lb["location"] = loc
    leaderboards.append(lb)
    if use_tune_data:
        tuning_data[tuning_data["location"] == loc]["y"].plot()
```

```
                     model  score_test  score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0      WeightedEnsemble_L2  -19.784612 -18.259556        1.190219       2.577594
172.023397                 0.003609                0.000605           0.406425
2       True          12
1    RandomForestMSE_BAG_L1  -20.506324 -19.037859        0.465188       1.088299
9.389158                 0.465188                1.088299           9.389158
1       True           5
2     NeuralNetTorch_BAG_L1  -20.636468 -19.420133        0.239440       0.374354
160.612212                 0.239440                0.374354         160.612212
1       True          10
3      ExtraTreesMSE_BAG_L1  -20.913315 -19.183511        0.481981       1.114335
1.615602                 0.481981                1.114335           1.615602
1       True           7
4          CatBoost_BAG_L2  -21.861565 -20.399400       13.679230      80.149340
671.681277                 0.042715                0.061000           9.000148
2       True          16
5      WeightedEnsemble_L3  -21.974210 -20.269160       14.537227      81.824679
774.139362                 0.003396                0.000671           0.340890
3       True          22
6      LightGBMLarge_BAG_L1  -22.002466 -19.942043        4.882097      22.486938
```

```
102.458817                 4.882097                22.486938         102.458817
1       True          11
7     ExtraTreesMSE_BAG_L2  -22.133535 -20.501681       14.126836       81.223771
664.699806                 0.490320                 1.135432           2.018677
2       True          17
8        LightGBMXT_BAG_L2  -22.159803 -20.556477       13.735883       80.388342
666.766527                 0.099368                 0.300002           4.085398
2       True          13
9          LightGBM_BAG_L2  -22.295762 -20.662814       13.682784       80.209733
664.911675                 0.046268                 0.121393           2.230546
2       True          14
10   NeuralNetTorch_BAG_L2  -22.335899 -20.515756       14.000797       80.627576
762.779648                 0.364281                 0.539237         100.098518
2       True          20
11          XGBoost_BAG_L2  -22.358626 -20.722484       13.722353       80.200207
665.610823                 0.085837                 0.111868           2.929693
2       True          19
12    LightGBMLarge_BAG_L2  -22.394402 -20.537913       14.433328       81.745611
729.687910                 0.796812                 1.657271          67.006781
2       True          21
13  RandomForestMSE_BAG_L2  -22.512068 -20.789223       14.114488       81.205569
676.115350                 0.477973                 1.117229          13.434220
2       True          15
14  NeuralNetFastAI_BAG_L2  -22.805315 -20.989446       14.292718       80.574287
698.871838                 0.656202                 0.485947          36.190709
2       True          18
15          XGBoost_BAG_L1  -23.200676 -21.473426        2.839509       20.574657
92.570059                 2.839509                20.574657          92.570059
1       True           9
16         LightGBM_BAG_L1  -23.361430 -21.563789        1.922492       19.250667
32.670249                 1.922492                19.250667          32.670249
1       True           4
17  NeuralNetFastAI_BAG_L1  -23.810966 -22.354976        0.641081        0.443902
35.495719                 0.641081                 0.443902          35.495719
1       True           8
18         CatBoost_BAG_L1  -24.726350 -22.578804        0.107755        0.102869
197.158424                 0.107755                 0.102869         197.158424
1       True           6
19       LightGBMXT_BAG_L1  -24.872298 -22.869948        1.988891       13.900755
30.658652                 1.988891                13.900755          30.658652
1       True           3
20   KNeighborsDist_BAG_L1  -27.173093 -26.226721        0.030431        0.376924
0.026137                 0.030431                 0.376924           0.026137
1       True           2
21   KNeighborsUnif_BAG_L1  -27.275008 -25.929619        0.037650        0.374639
0.026100                 0.037650                 0.374639           0.026100
1       True           1
```



[13]:
```
loc = "C"
predictors[2] = fit_predictor_for_location(loc)
```

```
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=1, num_bag_folds=8,
num_bag_sets=1
```

```
Training model for location C…
```

```
Beginning AutoGluon training …
AutoGluon will save models to "AutogluonModels/submission_94_C/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   229.73 GB / 315.93 GB (72.7%)
Train Data Rows:    23141
Train Data Columns: 38
Tuning Data Rows:    1400
Tuning Data Columns: 38
Label Column: y
Preprocessing data …
```

```
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and label-values can't be converted to int).
        Label info (max, min, mean, stddev): (999.6, 0.0, 82.28807, 171.35018)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    128643.73 MB
        Train Data (Original)  Memory Usage: 8.69 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 35 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 35 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        36 features in original data used to generate 36 features in processed
data.
        Train Data (Processed) Memory Usage: 6.9 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.15s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
```

```
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
AutoGluon will fit 2 stack levels (L1 to L2) …
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 …
        -23.8716         = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        0.83s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 …
        -24.1578         = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        0.25s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -21.6402         = Validation score   (-mean_absolute_error)
        28.93s   = Training   runtime
        14.97s   = Validation runtime
Fitting model: LightGBM_BAG_L1 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -21.8184         = Validation score   (-mean_absolute_error)
        29.96s   = Training   runtime
        8.21s    = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 …
```

```
	-21.1265	 = Validation score   (-mean_absolute_error)
	5.06s	 = Training   runtime
	0.74s	 = Validation runtime
Fitting model: CatBoost_BAG_L1 …
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-21.4958	 = Validation score   (-mean_absolute_error)
	191.58s	 = Training   runtime
	0.1s	 = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 …
	-20.8786	 = Validation score   (-mean_absolute_error)
	1.03s	 = Training   runtime
	0.76s	 = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 …
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-19.7355	 = Validation score   (-mean_absolute_error)
	29.32s	 = Training   runtime
	0.4s	 = Validation runtime
Fitting model: XGBoost_BAG_L1 …
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-21.5175	 = Validation score   (-mean_absolute_error)
	51.17s	 = Training   runtime
	3.23s	 = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 …
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-21.1018	 = Validation score   (-mean_absolute_error)
	64.76s	 = Training   runtime
	0.33s	 = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 …
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-21.2795	 = Validation score   (-mean_absolute_error)
	88.73s	 = Training   runtime
	8.6s	 = Validation runtime
Fitting model: WeightedEnsemble_L2 …
	-19.2408	 = Validation score   (-mean_absolute_error)
	0.44s	 = Training   runtime
	0.0s	 = Validation runtime
Fitting 9 L2 models …
Fitting model: LightGBMXT_BAG_L2 …
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-22.2002	 = Validation score   (-mean_absolute_error)
	2.27s	 = Training   runtime
	0.14s	 = Validation runtime
```

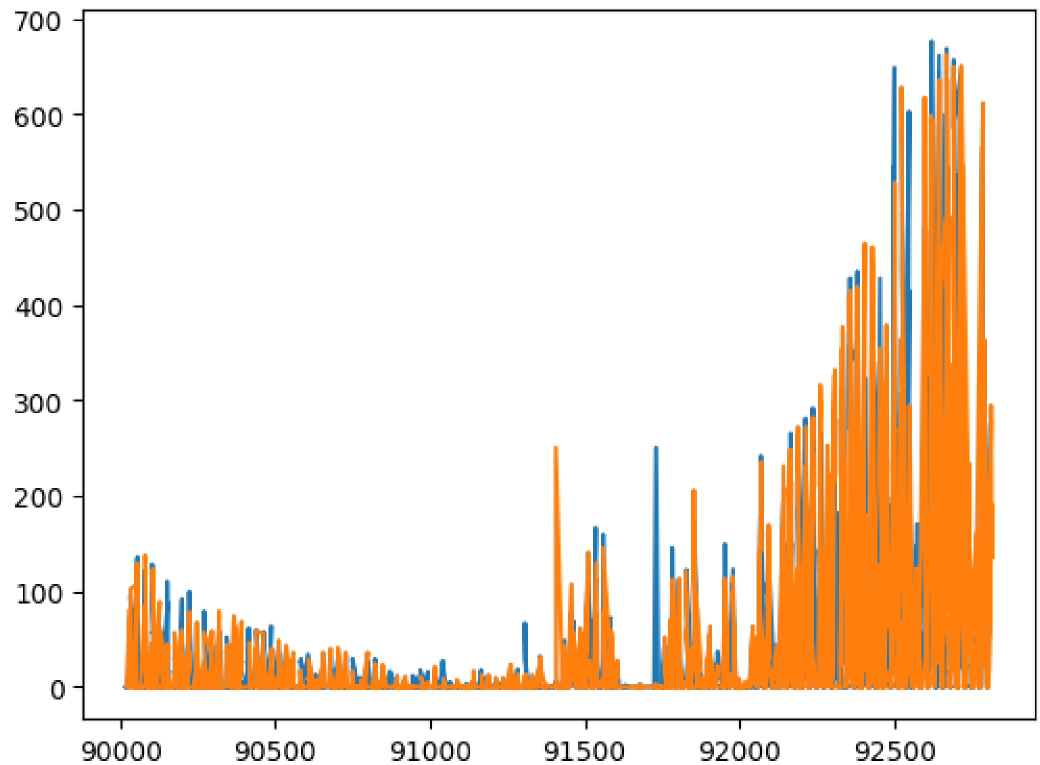
```
Fitting model: LightGBM_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -21.4717         = Validation score   (-mean_absolute_error)
        1.78s    = Training   runtime
        0.07s    = Validation runtime
Fitting model: RandomForestMSE_BAG_L2 …
        -21.5425         = Validation score   (-mean_absolute_error)
        8.23s    = Training   runtime
        0.79s    = Validation runtime
Fitting model: CatBoost_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -21.399  = Validation score   (-mean_absolute_error)
        6.9s     = Training   runtime
        0.05s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L2 …
        -21.3696         = Validation score   (-mean_absolute_error)
        1.31s    = Training   runtime
        0.8s     = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -21.3348         = Validation score   (-mean_absolute_error)
        29.79s   = Training   runtime
        0.39s    = Validation runtime
Fitting model: XGBoost_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -20.7431         = Validation score   (-mean_absolute_error)
        2.95s    = Training   runtime
        0.1s     = Validation runtime
Fitting model: NeuralNetTorch_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -21.4385         = Validation score   (-mean_absolute_error)
        51.7s    = Training   runtime
        0.46s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L2 …
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -21.5175         = Validation score   (-mean_absolute_error)
        5.92s    = Training   runtime
        0.19s    = Validation runtime
Fitting model: WeightedEnsemble_L3 …
        -20.7431         = Validation score   (-mean_absolute_error)
        0.34s    = Training   runtime
        0.0s     = Validation runtime
```

```
AutoGluon training complete, total runtime = 645.08s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_94_C/")
Evaluation: mean_absolute_error on test data: -19.739339913797004
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -19.739339913797004,
    "root_mean_squared_error": -48.6566638550214,
    "mean_squared_error": -2367.470937500546,
    "r2": 0.7684410045169413,
    "pearsonr": 0.9031989428540793,
    "median_absolute_error": -0.988118712902069
}

Evaluation on test data:
-19.739339913797004
```

[14]:
```python
if use_test_data:
    lb = predictors[2].leaderboard(test_data[test_data["location"] == loc])
    test_data[test_data["location"] == loc]["y"].plot()
    lb["location"] = loc
    leaderboards.append(lb)
    if use_tune_data:
        tuning_data[tuning_data["location"] == loc]["y"].plot()
```

```
                     model  score_test  score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0      WeightedEnsemble_L2  -19.739340 -19.240758        1.042274       2.310131
95.566443                 0.003901                0.000656           0.435457
2        True         12
1   NeuralNetFastAI_BAG_L1  -19.785365 -19.735470        0.500163       0.399011
29.319467                 0.500163                0.399011          29.319467
1        True          8
2    NeuralNetTorch_BAG_L1  -19.836243 -21.101790        0.214143       0.325194
64.757850                 0.214143                0.325194          64.757850
1        True         10
3    NeuralNetTorch_BAG_L2  -20.786542 -21.438521       11.972856      38.874038
542.276094                 0.328881                0.455749           51.700819
2        True         20
4     ExtraTreesMSE_BAG_L1  -21.032200 -20.878620        0.300408       0.756305
1.029495                 0.300408                0.756305           1.029495
1        True          7
5   RandomForestMSE_BAG_L1  -21.035793 -21.126537        0.285563       0.743042
5.057846                 0.285563                0.743042           5.057846
1        True          5
```

```
6           XGBoost_BAG_L2  -21.092990 -20.743136       11.723059      38.515971
493.527910                 0.079084                0.097682          2.952635
2       True          19
7      WeightedEnsemble_L3  -21.092990 -20.743136       11.724888      38.516615
493.869780                 0.001828                0.000645          0.341869
3       True          22
8           XGBoost_BAG_L1  -21.172547 -21.517455        1.273292       3.230571
51.172070                 1.273292                3.230571         51.172070
1       True           9
9     ExtraTreesMSE_BAG_L2  -21.247823 -21.369577       11.932498      39.216379
491.883744                 0.288523                0.798090          1.308468
2       True          17
10         CatBoost_BAG_L2  -21.290812 -21.399022       11.685564      38.470409
497.472901                 0.041589                0.052120          6.897625
2       True          16
11         CatBoost_BAG_L1  -21.322932 -21.495798        0.100098       0.103473
191.576589                 0.100098                0.103473        191.576589
1       True           6
12    LightGBMLarge_BAG_L1  -21.417781 -21.279517        4.108705       8.595093
88.725821                 4.108705                8.595093         88.725821
1       True          11
13  NeuralNetFastAI_BAG_L2  -21.440529 -21.334762       12.154256      38.805320
520.370076                 0.510281                0.387030         29.794801
2       True          18
14         LightGBM_BAG_L2  -21.474505 -21.471699       11.679493      38.490852
492.359499                 0.035518                0.072562          1.784223
2       True          14
15  RandomForestMSE_BAG_L2  -21.557714 -21.542477       11.923722      39.208714
498.800313                 0.279747                0.790424          8.225037
2       True          15
16       LightGBMXT_BAG_L1  -21.703644 -21.640242        2.594915      14.972013
28.927319                 2.594915               14.972013         28.927319
1       True           3
17         LightGBM_BAG_L1  -21.708364 -21.818396        2.221903       8.213923
29.960548                 2.221903                8.213923         29.960548
1       True           4
18    LightGBMLarge_BAG_L2  -21.710290 -21.517539       11.845297      38.604977
496.496011                 0.201322                0.186688          5.920736
2       True          21
19       LightGBMXT_BAG_L2  -21.886947 -22.200213       11.707611      38.556420
492.845138                 0.063636                0.138131          2.269862
2       True          13
20   KNeighborsUnif_BAG_L1  -23.769434 -23.871633        0.023660       0.828964
0.024173                 0.023660                0.828964          0.024173
1       True           1
21   KNeighborsDist_BAG_L1  -23.799712 -24.157808        0.021125       0.250699
0.024097                 0.021125                0.250699          0.024097
1       True           2
```



[15]:
```
# save leaderboards to csv
pd.concat(leaderboards).to_csv(f"leaderboards/{new_filename}.csv")
```

3 Submit

[16]:
```
import pandas as pd
import matplotlib.pyplot as plt

train_data_with_dates = TabularDataset('X_train_raw.csv')
train_data_with_dates["ds"] = pd.to_datetime(train_data_with_dates["ds"])

test_data = TabularDataset('X_test_raw.csv')
test_data["ds"] = pd.to_datetime(test_data["ds"])
#test_data
```

```
Loaded data from: X_train_raw.csv | Columns = 40 / 40 | Rows = 92945 -> 92945
Loaded data from: X_test_raw.csv | Columns = 39 / 39 | Rows = 4608 -> 4608
```

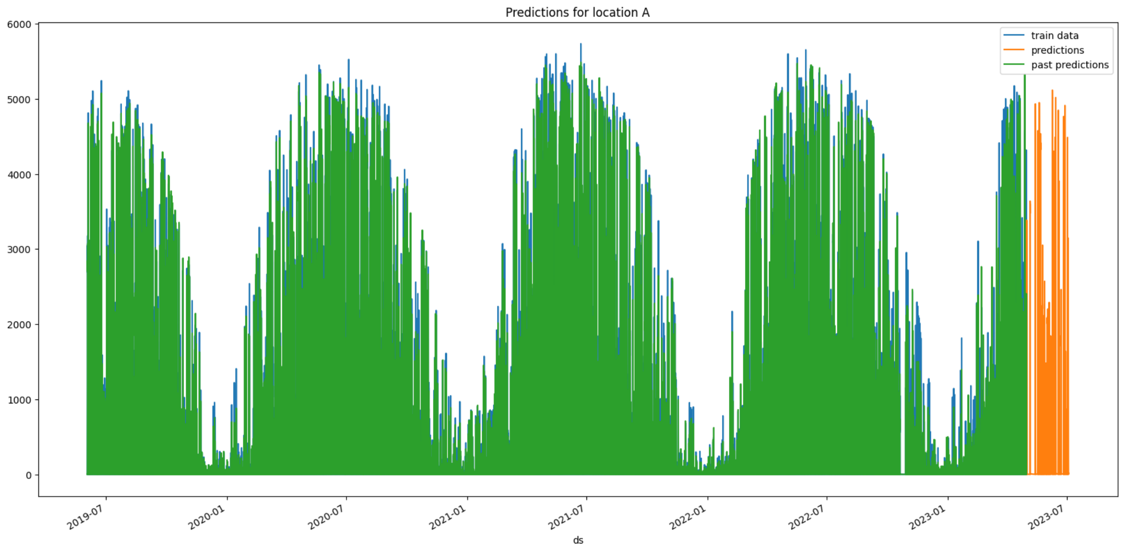
[17]:
```
test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
```

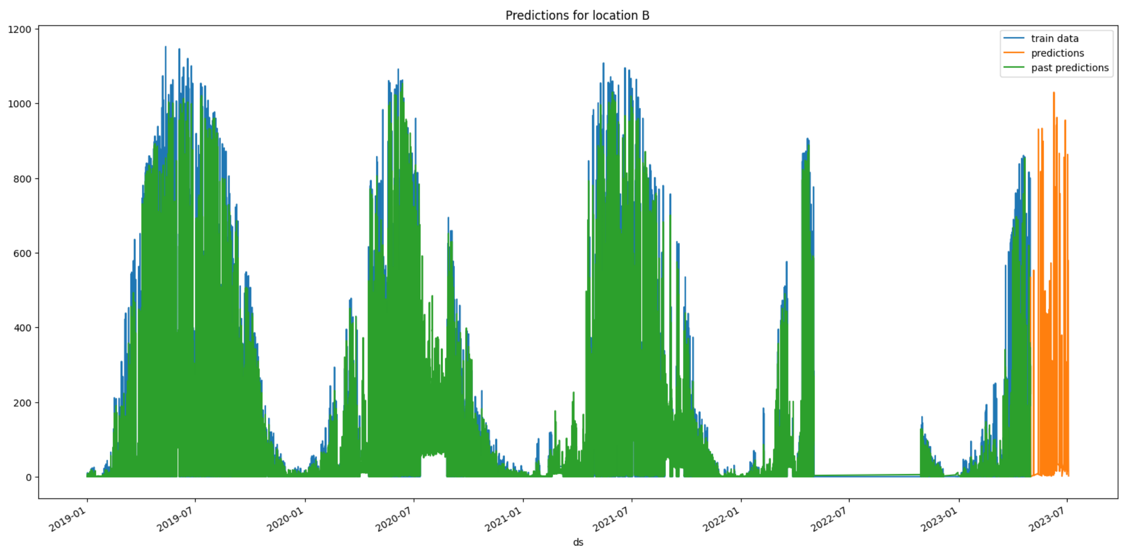
```
test_data_merged = pd.merge(test_data, test_ids, how="inner", right_on=["time",␣
↪"location"], left_on=["ds", "location"])

#test_data_merged
```

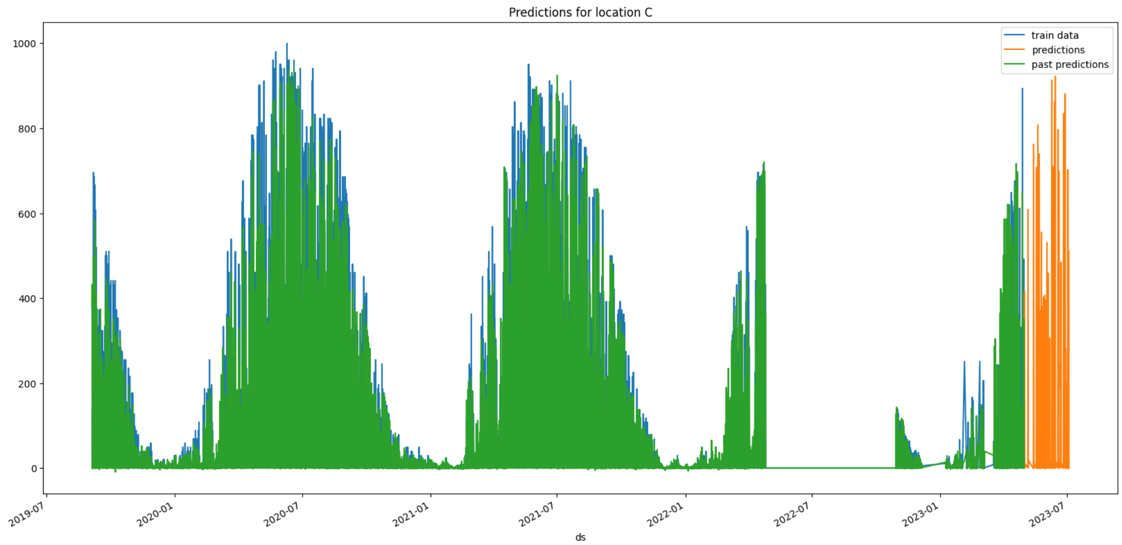
Loaded data from: test.csv | Columns = 4 / 4 | Rows = 2160 -> 2160

[18]:
```
# predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
for loc, group in test_data.groupby('location'):
    i = location_map[loc]
    subset = test_data_merged[test_data_merged["location"] == loc].
↪reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

    # get past predictions
    past_pred = predictors[i].
↪predict(train_data_with_dates[train_data_with_dates["location"] == loc])
    train_data_with_dates.loc[train_data_with_dates["location"] == loc,␣
↪"prediction"] = past_pred
```

[19]:
```
# plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
↪y='y', ax=ax, label="train data")

    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # plot past predictions
    train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
↪y='prediction', ax=ax, label="past predictions")

    # title
    ax.set_title(f"Predictions for location {loc}")
```


Predictions for location A
train data
predictions
past predictions
6000
5000
4000
3000
2000
1000
0
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
ds


Predictions for location B
train data
predictions
past predictions
1200
1000
800
600
400
200
0
2019-01
2019-07
2020-01
2020-07
2021-01
2021-07
2022-01
2022-07
2023-01
2023-07
ds



[20]:
```
# concatenate predictions
submissions_df = pd.concat(predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

[20]:

	id	prediction
0	0	0.388048
1	1	0.393996
2	2	0.383476
3	3	56.945789
4	4	347.566467
..	…	…
715	2155	46.367020
716	2156	14.130499
717	2157	1.687788
718	2158	0.084798
719	2159	18.746828

[2160 rows x 2 columns]

[21]:
```
# Save the submission DataFrame to submissions folder, create new name based on
↪last submission, format is submission_<last_submission_number + 1>.csv

# Save the submission
print(f"Saving submission to submissions/{new_filename}.csv")
submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),
↪index=False)
print("jall1a")
```

```
Saving submission to submissions/submission_94.csv
jall1a
```

[22]:
```python
# save this running notebook
from IPython.display import display, Javascript
import time

# hei123

display(Javascript("IPython.notebook.save_checkpoint();"))

time.sleep(3)
```

```
<IPython.core.display.Javascript object>
```

[23]:
```python
# save this notebook to submissions folder
import subprocess
import os
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
 ↪ipynb"])
```

```
[NbConvertApp] Converting notebook autogluon_each_location.ipynb to pdf
/opt/conda/lib/python3.10/site-packages/nbconvert/utils/pandoc.py:51:
RuntimeWarning: You are using an unsupported version of pandoc (2.9.2.1).
Your version must be at least (2.14.2) but less than (4.0.0).
Refer to https://pandoc.org/installing.html.
Continuing with doubts…
  check_pandoc_version()
[NbConvertApp] Support files will be in notebook_pdfs/submission_94_files/
[NbConvertApp] Making directory
./notebook_pdfs/submission_94_files/notebook_pdfs
[NbConvertApp] Writing 144295 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 210326 bytes to notebook_pdfs/submission_94.pdf
```

[23]:
```
CompletedProcess(args=['jupyter', 'nbconvert', '--to', 'pdf', '--output',
'notebook_pdfs/submission_94.pdf', 'autogluon_each_location.ipynb'],
returncode=0)
```

[24]:
```python
# feature importance
location="A"
split_time = pd.Timestamp("2022-10-28 22:00:00")
estimated = train_data_with_dates[train_data_with_dates["ds"] >= split_time]
```

```
estimated = estimated[estimated["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=estimated,␣
↪time_limit=60*10)
```

```
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'location', 'prediction']
Computing feature importance via permutation shuffling for 36 features using
4392 rows with 10 shuffle sets… Time limit: 600s…
        6164.29s        = Expected runtime (616.43s per shuffle set)
        349.2s  = Actual runtime (Completed 1 of 10 shuffle sets) (Early
stopping due to lack of time…)
```

[24]:

	importance	stddev	p_value	n	p99_high \
direct_rad_1h:J	1.661192e+02	NaN	NaN	1	NaN
diffuse_rad_1h:J	3.306991e+01	NaN	NaN	1	NaN
sun_azimuth:d	3.023113e+01	NaN	NaN	1	NaN
clear_sky_energy_1h:J	2.874515e+01	NaN	NaN	1	NaN
clear_sky_rad:W	2.338161e+01	NaN	NaN	1	NaN
direct_rad:W	1.696939e+01	NaN	NaN	1	NaN
diffuse_rad:W	1.157282e+01	NaN	NaN	1	NaN
total_cloud_cover:p	9.132186e+00	NaN	NaN	1	NaN
cloud_base_agl:m	5.896452e+00	NaN	NaN	1	NaN
relative_humidity_1000hPa:p	4.958079e+00	NaN	NaN	1	NaN
snow_water:kgm2	4.532055e+00	NaN	NaN	1	NaN
effective_cloud_cover:p	4.454154e+00	NaN	NaN	1	NaN
fresh_snow_6h:cm	3.940948e+00	NaN	NaN	1	NaN
sun_elevation:d	3.692114e+00	NaN	NaN	1	NaN
wind_speed_10m:ms	3.464202e+00	NaN	NaN	1	NaN
visibility:m	3.348638e+00	NaN	NaN	1	NaN
t_1000hPa:K	3.316352e+00	NaN	NaN	1	NaN
is_in_shadow:idx	2.676490e+00	NaN	NaN	1	NaN
precip_5min:mm	1.691140e+00	NaN	NaN	1	NaN
sfc_pressure:hPa	1.440330e+00	NaN	NaN	1	NaN
super_cooled_liquid_water:kgm2	1.249810e+00	NaN	NaN	1	NaN
is_day:idx	1.040715e+00	NaN	NaN	1	NaN
snow_melt_10min:mm	1.026113e+00	NaN	NaN	1	NaN
precip_type_5min:idx	1.003837e+00	NaN	NaN	1	NaN
fresh_snow_3h:cm	8.393493e-01	NaN	NaN	1	NaN
ceiling_height_agl:m	5.453121e-01	NaN	NaN	1	NaN
pressure_100m:hPa	2.547247e-01	NaN	NaN	1	NaN
air_density_2m:kgm3	1.916241e-01	NaN	NaN	1	NaN
snow_depth:cm	1.884514e-01	NaN	NaN	1	NaN
is_estimated	5.189622e-08	NaN	NaN	1	NaN
rain_water:kgm2	-2.983781e-02	NaN	NaN	1	NaN
fresh_snow_1h:cm	-9.135414e-02	NaN	NaN	1	NaN
pressure_50m:hPa	-3.242515e-01	NaN	NaN	1	NaN
msl_pressure:hPa	-3.661329e-01	NaN	NaN	1	NaN

```
dew_point_2m:K                -1.109279e+00     NaN      NaN  1        NaN
absolute_humidity_2m:gm3      -1.179244e+00     NaN      NaN  1        NaN

                               p99_low
direct_rad_1h:J                    NaN
diffuse_rad_1h:J                   NaN
sun_azimuth:d                      NaN
clear_sky_energy_1h:J              NaN
clear_sky_rad:W                    NaN
direct_rad:W                       NaN
diffuse_rad:W                      NaN
total_cloud_cover:p                NaN
cloud_base_agl:m                   NaN
relative_humidity_1000hPa:p        NaN
snow_water:kgm2                    NaN
effective_cloud_cover:p            NaN
fresh_snow_6h:cm                   NaN
sun_elevation:d                    NaN
wind_speed_10m:ms                  NaN
visibility:m                       NaN
t_1000hPa:K                        NaN
is_in_shadow:idx                   NaN
precip_5min:mm                     NaN
sfc_pressure:hPa                   NaN
super_cooled_liquid_water:kgm2     NaN
is_day:idx                         NaN
snow_melt_10min:mm                 NaN
precip_type_5min:idx               NaN
fresh_snow_3h:cm                   NaN
ceiling_height_agl:m               NaN
pressure_100m:hPa                  NaN
air_density_2m:kgm3                NaN
snow_depth:cm                      NaN
is_estimated                       NaN
rain_water:kgm2                    NaN
fresh_snow_1h:cm                   NaN
pressure_50m:hPa                   NaN
msl_pressure:hPa                   NaN
dew_point_2m:K                     NaN
absolute_humidity_2m:gm3           NaN
```

[ ]:
```
# feature importance
observed = train_data_with_dates[train_data_with_dates["ds"] < split_time]
observed = observed[observed["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=observed, time_limit=60*10)
```

```
These features in provided data are not utilized by the predictor and will be
```

```
ignored: ['ds', 'elevation:m', 'location', 'prediction']
Computing feature importance via permutation shuffling for 36 features using
5000 rows with 10 shuffle sets… Time limit: 600s…
        8118.65s        = Expected runtime (811.87s per shuffle set)
```

[ ]:
```
display(Javascript("IPython.notebook.save_checkpoint();"))
time.sleep(3)

subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}_with_feature_importance.pdf"),␣
 ↪"autogluon_each_location.ipynb"])
```

[ ]:
```
# import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
#         result = subprocess.check_output(['git', '-C', directory] + command,␣
 ↪stderr=subprocess.STDOUT)
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
 ↪strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."

# execute_git_command(git_repo_path, ['config', 'user.email',␣
 ↪'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is␣
 ↪not None else 'Henrik eller Jørgen'])

# branch_name = new_filename

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

# commit_msg = "run result"

# execute_git_command(git_repo_path, ['checkout', '-b',branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m',commit_msg])

# # Push to remote
```

```
# output, success = execute_git_command(git_repo_path, ['push',␣
↪'origin',branch_name])

# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
#     execute_git_command(git_repo_path, ['push', '--set-upstream',␣
↪'origin',branch_name])
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

# execute_git_command(git_repo_path, ['checkout', 'main'])
```